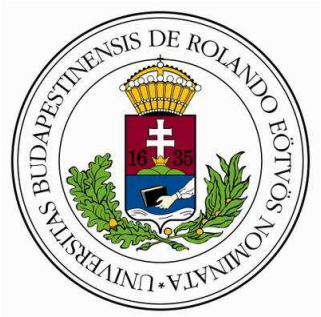


Eötvös Loránd Tudományegyetem
Informatikai Kar
Információs rendszerek tanszék

Kooperatív érzékelés járműkommunikációs környezetben

Dr. Laki Sándor
Adjunktus
Dr. Edelmayer András
Tudámyos Tanácsadó

Lövei Péter
Programtervező Informatikus BSc

Budapest, 2019

Tartalomjegyzék

1. Bevezető

1.1. Témamegjelölés

A grafikus processzorok ugrásszerű fejlődésének köszönhetően megjelentek - az okos város illetve az önvezető autó témakörébe tartozó - olyan, a peremhálózatára kitelepített, vagy IoT eszközök, és azokon implementált speciális algoritmusok, melyek lehetővé teszik a valós idejű képfeldolgozást. Ezek a megvalósítások a kamera által szolgáltatott raw-data streamet feldolgozzák, majd különböző adatfúziós lépéseken keresztül átalakítják annak érdekében, hogy a kamera képben megjelenő objektumokat, illetve azok mozgását detektálni tudjuk, majd ezekhez az absztrakt entitásokhoz köthető kinyert információkat különböző technológiák segítségével (esetemben járműkommunikációs módszerekkel) megoszthassák a környezetükben lévő hasonló célú eszközökkel. Az ilyen rendszerek aztán képesek lesznek olyan közlekedési szituációkhoz köthető gyors, kollektív döntéseket meghozni a jelenleg rendelkezésre álló mesterséges intelligencia módszerek alkalmazásával, amelyekre a hagyományos informatikai rendszerek eddig nem voltak képesek.

Szakdolgozatomban ilyen eszközök konfigurálásval, a rajtuk futó algoritmusok implementálásával és vizsgálatával, kommunikációjának lehetőségeivel illetve performanciabeli összehasonlításával foglalkozom, amelyhez általam kifejlesztett metrikákat alkalmazok. Utóbbi alapjául egyrészt adatokat gyűjtök, másrészt egy erre a célra kijelölt programot készítek. Végül a kinyert eredményeket diagrammok formájában teszem összehasonlíthatóvá.

Már régóta terveztem egy nagyobb python projekt megvalósítását, ráadásul a munkahelyemen igény támadt különböző modellek összemérésére a késlekedési időt szem előtt tartva, ezért kaptam a lehetőségen és elvállaltam a feladatot. A cél, a modellek egy performacia- és pontosságbeli összehasonlítása egy speciális GPU-n.

A feladat megoldására egy olyan program megvalósítását tűztem ki célul, mely egyrészt robosztus másrészt hatékonyan tudja végezni a kiértékelést. Harmadrészt fontos szempont volt számomra az is, hogy a lehető leggenerikusabb legyen, hiszen szerettem volna, ha más cégek, akik ugyancsak objektumfelismeréssel (is) foglalkoznak kis módosítások segítségével ki tudják értékelni tetszőlegesen nyílt forráskódú felismerőjüket a saját környezetükben, saját metrikáik segítségével. A feladat során először a különböző képeket és annotációkat kell beolvasni, majd az objektumfelismerőket. Ezt követően egyesével ki kell értékelni a képeket a metrikák segítségével. Az eredményeket összegezni kell, majd ez alapján lehet meghatározni a sorrendjüket, illetve ábrázolni diagrammokon.

A jelen szakdolgozatom munkája, eredménye a Commsignia ITS-RS4 RSU(Road side unit)-nak ad információt. Ezek az eszközök a V2X(Vehicle-to-everything)-es

rendszerben nyújtanak számos funkcionalitást, mint például szenzorok fuzionálása vagy gyalogosok felügyelete.

1. ábra. Commsignia ITS-RS4 RSU

1.2. Köszönetnyílvánítás

Szeretném megköszönni a Commsignia-nak a remek lehetőséget, hogy ilyen izgalmas projekttel foglalkozhattam. A dolgozat elkészítése során, mind a környezet, mind a hardverbeli támogatás kifogásolhatatlan volt, jelentősen hozzájárulva ezzel a hatékony tervezéshez, fejlesztéshez. Ezen felül fontos kiemelni a pénzügyi támogatásukat is, hiszen önerőből valószínűleg nem tudtam volna vásárolni egy ilyen eszközt.

Továbbá szeretném megköszönni családtagjaimnak és barátaimnak a szellemi támogatást. Nélkülük nem tudtam volna ilyen eredményesen elvégezni a feladatot.

Lövei Péter

2. Felhasználói dokumentáció

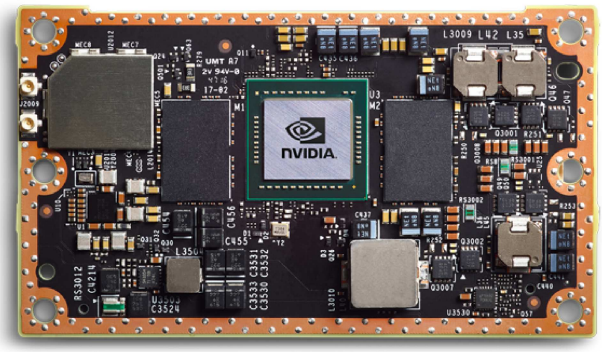
2.1. A feladat leírása

A feladat egy olyan megoldást vár el, mely képes képeket és annotációkat beolvasni (utóbbit szöveges fájlból), majd a képen különböző modellek detekciójának eredményeit összehasonlítani az annotációkban található metaadaatokkal. Ehhez az összehasonlításra különböző metrikákat kell definiálni, melyek képesek különböző szempontból megvizsgálni a detekciók helyességét. Az ilyen módon kiértékelt adatokat végül statisztikai módszerek segítségével aggregálni kell egyrészt egy sorrend felállításához, másrészt az eredmények vizualizáláshoz ezzel is hozzájárulva az a legalkalmasabb model megválasztásának döntéshozatalához.

A feladathoz tartozó fontos célkitűzés volt még a részemről az is, hogy legyen annyira általános a keretrendszer, képes legyen tetszőleges metrikákkal (és nem csak az általam definiáltakkal), nyílt forráskódú objektumfelismerőkkel, statisztikákkal működni, ezáltal elősegítve a különböző ilyen területtel foglalkozó cégek helyzetét. Hiszen a generikus megvalósításnak köszönhetően, kis módosítások után tudják mérni felismerőiket a saját paramétereik és környezetük alapján, amely jelentősen csökkentheti a számukra optimális model kiválasztására szánt időt.

A feladat egyik sajátossága, hogy mindezt egy olyan beépített eszközön kell megvalósítani, mely alapvetően csak egy lapkából áll. Az Nvidia által gyártott Jetson TX2-es module magas performanciára képes különösképpen interferencia esetében. Másrészről az ilyen objektumfelismerőknél az egyik legfontosabb aspektus a késlekedési idő, hiszen ez a paraméter jelentősen tudja befolyásolni, mennyire hatékony valós helyzetben, ahol az eredményekre azonnal szükség van.

2. ábra. Jetson TX2



2.2. Felhasznált módszerek

2.2.1. Programnyelv

A probléma megoldása során számos módszert felhasználtam. Ami a programozási nyelvet illeti, Python 3-t választottam, hiszen egyrészt már korábban is programoztam vele több alkalommal is. Másrészt számos olyan könyvtár használható vele könnyedén és hatékonyan melyek adatok vizualiációjához köthetőek, segítve a hatékony megértést. Végezetül véleményem szerint egy olyan nyelvről van szó, melynek kódja könnyen érthető, mely egy fontos szempont csapatmunka és vagy bővítés során.

2.2.2. Metrikák

A megvalósítás generikus a metrikákra nézve. A programnak tetszőlegesen megadható bármely olyan metrika család, mely detektált objektumokon és hozzájuk fűződő annotációkon van értelmezve. Az általam használt család tagjainak megválasztásánál fontos szerepet játszott a kísérletezés és az egyetemi tanulmányaim során összegyűjtött tudás, hiszen megítélésem szerint olyanokat választottam, melyek hatékonyan tudják mérni a felismerők pontosságát, performanciáját.

Mielőtt ismertetném az általam használt metrikákat, szeretnék kitérni a detektálás kiértékelésének átfogó összefoglalására. A detektálások minden egyes alkalommal egy-egy képen történtek, amelyekhez egy-egy annotáció társult.
Az egyes detektáló algoritmusok eltérő felépítésük ellenére megkapták a képet inputnak, és outputnak megadtak több, a felismerni vélt objektum(ok)hoz tartozó információt, mint például:

- annak helyzetét a 2D pixeltérben megadó téglalap bal felső, és jobb alsó x,y koordinátája,
- milyen objektumnak gondolja (pl.: ember, kutya, asztal)
- ezt mekkora pontossággal állítja[1]

$f_{det}\colon \mathbb{R}^2 \to (\mathcal{P} \times \mathcal{T} \times \mathcal{C})^k = \mathcal{D}; k \in \mathbb{N}, \mathcal{P} \subseteq \mathbb{R}^4, \mathcal{C} \subseteq [0,1]$
Ahol $\mathcal{P}$ a bal felső és jobb alsó koordináták sorban, $\mathcal{T}$ az objektum típusa, $\mathcal{C}$ a konfidenciaérték, $\mathbb{N}$ pedig az adott kép indexe. Így minden egyes képhez előállít egy prediktált eredmény. Ugyanakkor minden képhez tartozott egy annotáció is, mely ugyancsak leírta:

[1] A detektorok általában egy n-dimenziós valószínűségi vektorváltozót adnak vissza, melyben a valószínűségi értékek az egyes objektumtípusokhoz köthető. Én a kiértékelés során a legnagyobb konfidenciaértékűt hagytam meg, azok után, hogy egy előszűrést végeztem rajtuk, kidobva azokat a detekciókat ahol a maximális érték nem halad meg egy bizonyos küszöbszintet.

- az adott képen található objektumok valós 2D-s pozícióját
- annak típusát.

$f_{ann}\colon \mathbb{N} \to (\mathcal{P} \times \mathcal{T})^k = \mathcal{A}, k \in \mathbb{N}$ Ezáltal a metrikák során a két, képből az imént említett értékekbe képező függvény értékeit hasonlítottam össze. Az ily módon képzett metrikáim a következő formulával írhatók le általánosan: $f_{met}\colon f_{det} \times f_{ann} \to \mathbb{R}$

A következőkben a $\delta_t, \delta_p, \delta_T, \delta_c$ sorra azt jelölik, hogy az adott $\mathcal{P} \times \mathcal{T} \times \mathcal{C}$ rendezett hármas éppen melyik komponensével számolunk (kivéve δ_T-t, az a csúcspontok által kifeszített téglalapra utal). Az α esetében hasonló a helyzet, ugyanakkor az α^δ jelölést azt mutatja, hogy éppen az aktuális detekcióhoz legközelebbi annotációról van szó, ezt hivatott jelölni a felső index.

Ezekhez a következő függvényeket használtam:[2]

- Az átlagos euklideszi távolság

$$\overline{\sum_{\delta \in \mathcal{D}} ||\delta_p - \alpha_p^\delta||_1}, \quad \forall \alpha^\delta \in \mathcal{A} \tag{1}$$

- IOU
  Az egyes detekciók és a hozzájuk tartozó annotációk által kifeszített síkbeli területek metszetének is uniójának hányadosoknak az átlaga

$$\overline{\sum_{\delta \in \mathcal{D}} \frac{\delta_T \cap \alpha_T^\delta}{\delta_T \cup \alpha_T^\delta}}, \quad \forall \alpha^\delta \in \mathcal{A} \tag{2}$$

- Pontossági ráta
  A helyes detekciókhoz tartozó konfidencia értékek szummájának és a helytelenekhez tartozó értékek szummájának különbsége

$$\sum_{\substack{\delta \in \mathcal{D} \\ \delta_\tau = \alpha_\tau^\delta}} \delta_c^2 - \sum_{\substack{\delta \in \mathcal{D} \\ \delta_\tau \neq \alpha_\tau^\delta}} \delta_c^2, \quad \forall \alpha^\delta \in \mathcal{A} \tag{3}$$

- Helyességi ráta
  A helyes detekciók és a helytelenek számának aránya

$$\frac{|\{\delta \in \mathcal{D}, \delta_\tau = \alpha_\tau^\delta\}|}{|\{\delta \in \mathcal{D}, \delta_\tau \neq \alpha_\tau^\delta\}|}, \quad \forall \alpha^\delta \in \mathcal{A} \tag{4}$$

[2] Az összehasonlítások során mindig az az egyes detekciókhoz a legközelebbi (egyes norma szerint) annotációt vettem alapul.

- Detekció ideje
  A detekció megkezdése és a felismert objektumok visszaadása közti időkülönbség

2.2.3. Az eredmények kiértékelése

Amint lefuttatuk a kiértékelést az eredményeket metrikánként először rendszerezem. Ehhez különböző, torzítatlan statisztikákat használok. Ezek segítségével ugyanis különböző fontos információkhoz jutunk az egyes detektorokkal kapcsolatban. Fontos kiemelni, hogy nem azért használom ezeket a statisztikákat, mert a program futásának helyessége ezekre van korlátozódva, hanem pusztán szubjektív okokból.
A felhasznált statisztikák:

- átlag
- variancia
- minimum
- maximum
- terjedelem

Az aggregált eredményeket aztán egyrészt vizuálisan megjelenítem egy előszűrést követően, ahol az adott diagrammtípusnak (box, oszlop) megfelelően kiszűröm a kiugró értékeket, vagy normalizálom őket, hogy egy egységes koordináta rendszerbe kerüljenek a reprezentatívabb megjelenés érdekében.

Másrészt az aggregált eredményeken végzek egy összetett kiértékelést. Segítségével meghatározom a detektálás során a kinyert értékek alapján a detektorok közti hatékonysági sorrendet. A sorrend számításnál az alábbi szempontokat veszem figyelembe:

- Az adott metrikának mekkora súlya van
- Performancia szempontjából maximalizálandó vagy minimalizálandó (Míg idő és az átlagos euklideszi távolság minimalizálandó, addig az iou a helyességi és pontossági ráták maximalizálandók.)
- Az egyes statisztikák milyen eredményeket adtak

Ezek alapján a kiértékelés lépései:

1. Minden egyes metrikához kiszámítjuk az egyes detektorok súlyozott aggregált helyezését az alábbiak szerint:

(a) Minden egyes statisztikához hozzárendelünk egy performanciabeli rendezési relációt attól függően egyrészt mely statisztikáról van szó, másrészt milyen az adott statisztikánk. Itt a kisebb helyezés számít a jobbnak, tehát például egy míg a detekció idejénél minden statisztikát minimalizálni szeretnénk, addig a pontosságnál már mondjuk a minimum és átlag értékek maximalizálandók, míg a variancia továbbra is minimalizálandó a stabilitást mérve ezáltal (összeségében a variancia és a terjedelem minimalizálandó).

(b) Az egyes detektorok eredményét az adott statisztikán belül az előbbi reláció segítségével rendezzük

(c) A helyezéseket összeadjuk detektoronként

(d) Végül ezt az akkumulált eredményt súlyozzuk az alapján a fejlesztő milyen fontossági sorrendet adott a metrikánknak.

2. A kapott helyezeséket szummázuk detektoronként, majd rendezzük növekvő sorrendben

3. Az így kapott sorrendel előáll a performanciabeli sorrend is.

Legyen $\mathcal{F}$ a metrikák halmaza, $\mathcal{D}$ a detektoroké, λ_f az adott metrikához tartozó súly, $\mathcal{S}$ pedig legyen olyan a statisztikákhoz tartozó függvények halmaza, amely adott statisztikához és detektorhoz visszaadja a detektor adott statisztika szerinti helyezését.[3]
Ekkor a formula a legjobb detektor kiszámítására (a rendezés analóg módon történik):

$$d_{\text{legjobb}} = \min_{d \in D} \left\{ \sum_{f \in \mathcal{F}} \lambda_f \cdot \left(\sum_{s \in \mathcal{S}} s(d) \right) \right\} \tag{5}$$

2.3. A program használatához szükséges információk

A kiértékelőt Linux operációs rendszerről lehet használni ARM architektúrájú processzorral. Mivel a számításigény igen magas az ilyen detektorok esetében ezért elengedhetetlen a futtatáshoz minimum 8GB ram, és egy erős, dedikált videókártya, hiszen a megvalósítás erősen épít ezekre az erőforrásokra. Mivel a kiértékelés igénye erősen összefügg az eszközzel amelyen azt elkészítettem, így a hardver kihasználtság erősen optimalizálva lett az NVIDIA Jetson TX2-es boardra, ezért javasolt vele, vagy hozzá hasonló eszközzel használni, mely rendelkezik számos az NVIDIA

[3]Tehát $|\mathcal{S}|$ megegyezik a statisztikák számával

GPU-kra optimalizált könyvtárral mint például a TensorRT. A programot parancssorból lehet használni különböző paraméterek megadásával, habár ezeknek vannak alapvető értékei.

2.3.1. Környezet

A program futtatásához a következő csomagok előzetes telepítésére van feltétlen szükség:

- Jetpack 3.3, mely az Nvidia Developer Programon keresztül érhető el.
- OpenCV 3.

Mivel a program nem jár felhasználói interakcióval ezért parancssorból lehet futtatni a

```
1    python3 main.py
```

parancs kiadásával.

2.3.2. Paraméterek

A program a következő paramétereket várja:

- A képeket tartalmazó mappa
- Az annotációkat tartalmazó mappa
- A metrikákat megvalósító osztály
- A detektorainkat buildelni szeretnénk-e
- Milyen konfidencia határ alattiakat dobjunk el

3. ábra. Illusztráció használatáról és paramétereiről

```
→ ai_benchmark git:(master) ✗ python3 main.py --help
DEBUG: Called __init__(<__main__.Main object at 0x7f45abb9e8>)
usage: main.py [-h] [--dir IMAGEDIR] [--annotation ANNOTATIONDIR]
               [--metrics METRICS] [--build] [--confidence CONFTH]

This script evalutes detectors based on given metrics

optional arguments:
  -h, --help            show this help message and exit
  --dir IMAGEDIR        The directory for the images
  --annotation ANNOTATIONDIR
                        The directory for the annotations
  --metrics METRICS     The metrics to be used
  --build               re-build TRT pb file (instead of usingthe previously
                        built version)
  --confidence CONFTH   confidence threshold [0.3]
```

2.4. Eredmények

A kiértékeléskor az alábbi detektorokat hasonlítottam össze:

- yolov3
- yolov3-tiny
- ssd-mobilenet-v1-coco
- ssd-inception-v2-coco
- ssdlite-mobilenet-v2-coco

Míg az első kettő az Darknet model, addig az utolsó három Google által kifejlesztett TensorRT által optimalizált model.

2.4.1. A kiválasztott detektorok indoklása

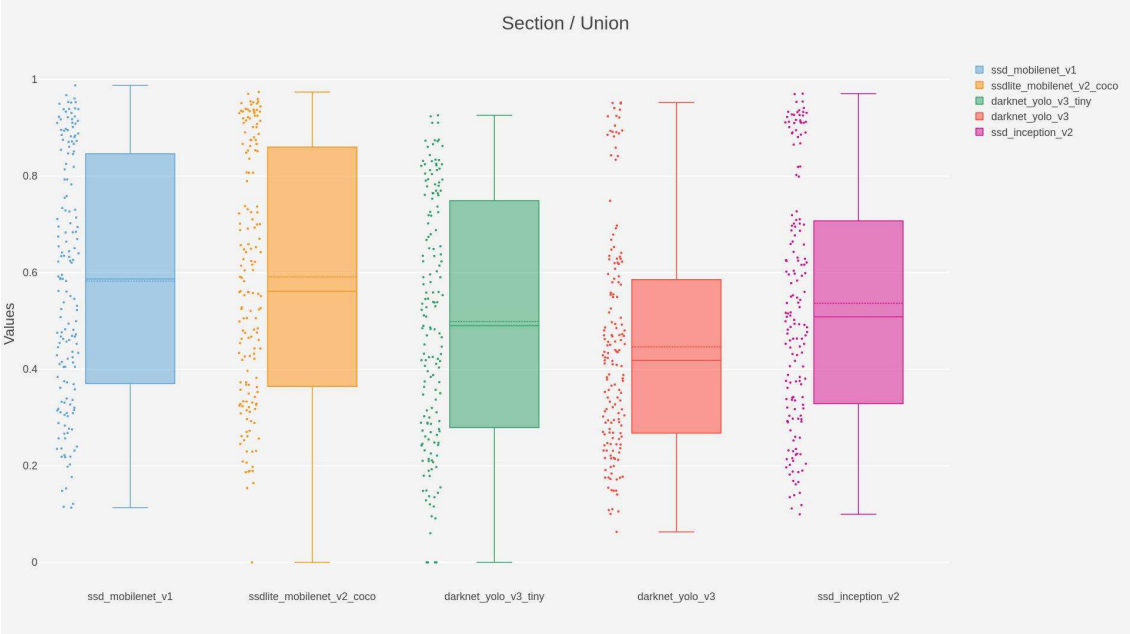
A használt detektorok kiválasztása mögött számos ok rejlik. Egyrészt a célkitűzés kapcsán nyílt forráskódú detektorokat akartam használni, másrészt csak olyan modelek jöttek szóba akik a kapott eszköz memóriájába (8GB) belefér, ugyanis számos model nem teljesíti ezt a követelményt.

2.4.2. Diagrammok

A 4, 5, 6, 7, 8, 9 és a 10 diagrammokon prezentálom az eredményeket. Ezeken a metrikák kiértékelése során keletkezett eredmények láthatók boxplot és oszlopdiagrammok formájában. A boxplotok esetében a pontok a boxok bal oldalán helyezkednek el, az egyes részek a következőképpen alakulnak felső vonal, box teteje, egyenes vonal, szagatott vonal, box alja, alsó vonal. A hozzájuk tartozó értékek pedig rendre: max{ max$\{x_i\}$, Q1 - 1.5 $\cdot$ IQR }, Q3, átlag, medián, Q1, min{ min$\{x_i\}$, Q3 + 1.5 $\cdot$ IQR }.

A 4. ábrán az IOU metrika eredményei láthatók. Az értékek 0 és 1 között mozognak, az 1 jelenti, hogy teljes fedésben vannak, 0 pedig, hogy nincs közös pontjuk a téglalapoknak. Láthatjuk, hogy hasonlóképpen teljesítettek, kivéve a yolo_v3-t, ami a legrosszabbul szerepelt.

4. ábra. IOU



A 5. ábrán az átlagos euklideszi távolság látható, mely minimalizálandó. A -20-as érték akkor keletkezett, amikor nem talált objektumot a detektor.

A 6. ábrán a detekciók ideje látható másodpercben kifejezve. Szemmel láthatóan a yolo_v3-nak tartott a legtovább az egyes képek kiértékelése, ugyanis minél kisebbek az értékek, annál jobb az eredmény.

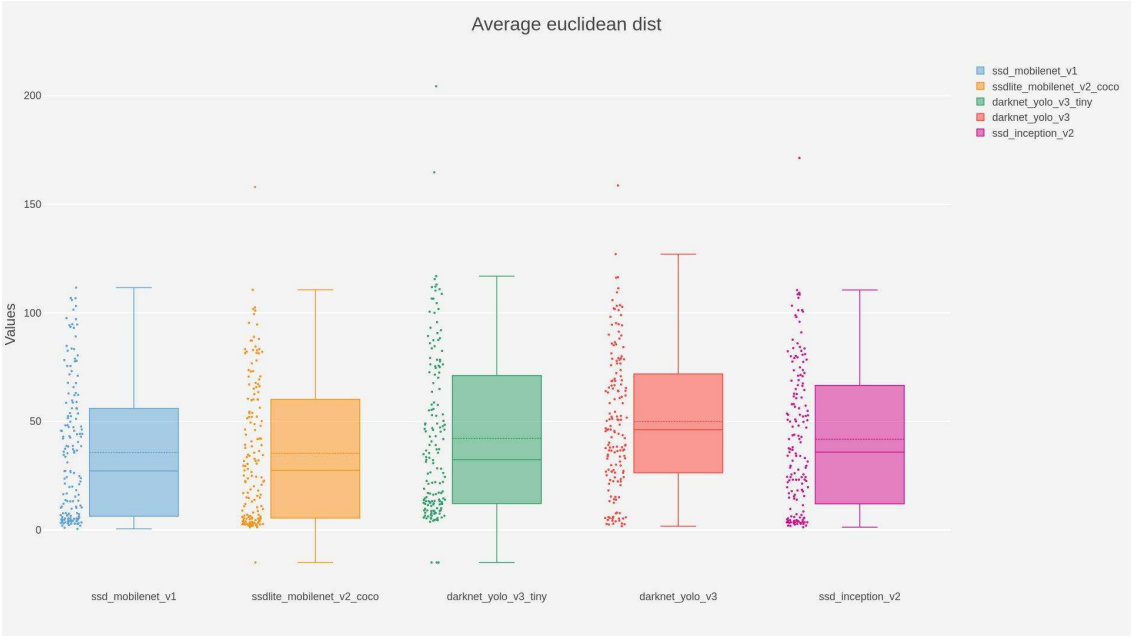
A 7. ábrán a helyességi ráta látható. Összeségében a detektorok egész hasonlóan teljesítettek.

A 8. ábrán a pontossági ráta látható. Mivel itt minél magasabb pontszám számít jó eredménynek, ezért alapvetően kijelenthető a yolo_v3 teljesített a legjobban, ugyanakkor nagy szóródással is.

A 9. ábrán a metrikákat összesítetten lehet látni. Ezt az egyes eredmények egy egységes, normalizált koordináta rendszerbe való transzformálása tette lehetővé.

A 10. ábrán a detektorok súlyozott összesített helyezései találhatók metrikánként. Felhívom a figyelmet, hogy minél alacsonyabb pontszámmal végzett egy detektor egy adott metrikán belül, ott annál jobbnak bizonyult. Ez alapján megfigyelhető, hogy az ssd_mobilenet_v1 model általánosságban a legjobban teljesített.

5. ábra. Átlagos euklideszi távolság



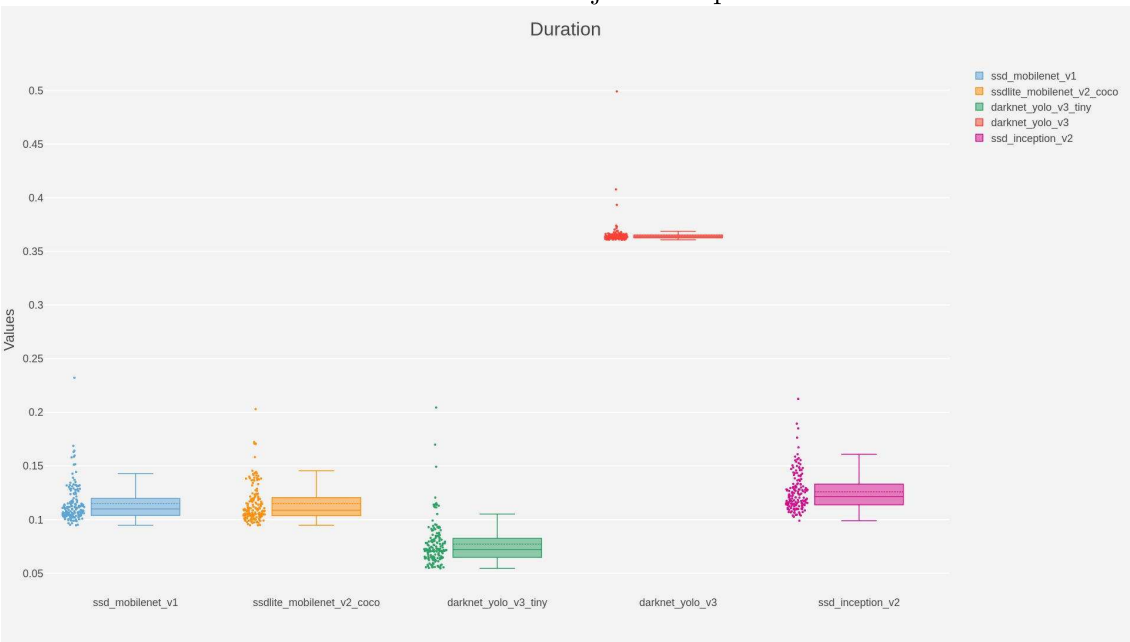
2.4.3. Sorrend

A végeredményt tehát az aggregált súlyozott helyezések adják. A kiértékeléssel elégedett vagyok, hiszen előzetes számításaim és teszteléseim szerint hasonló eredményre számítottam. Az ssd-mobilenet-v1 model teljesített a legjobban, míg a darknet-yolo-v3 a legrosszabbul. A sorrend a következő[4]:

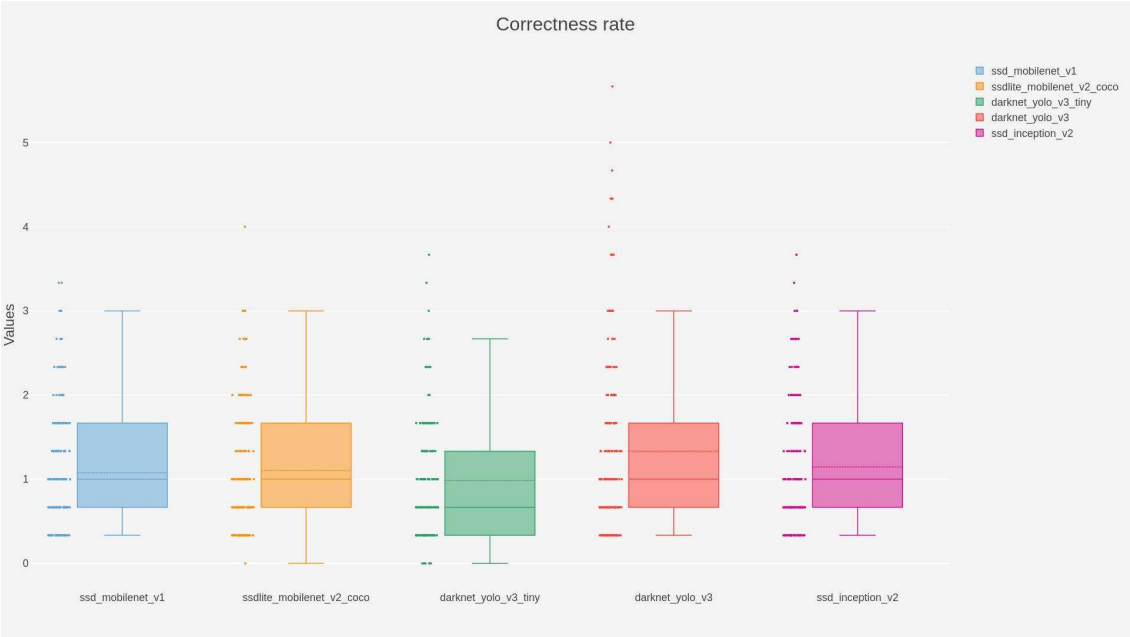
1. ssd-mobilenet-v1 131
2. ssdlite-mobilenet-v2 156
3. darknet-yolo-v3-tiny 169
4. ssd-inception-v2 181
5. darknet-yolo-v3 186

[4]Számos futtatásnak eredményképpen

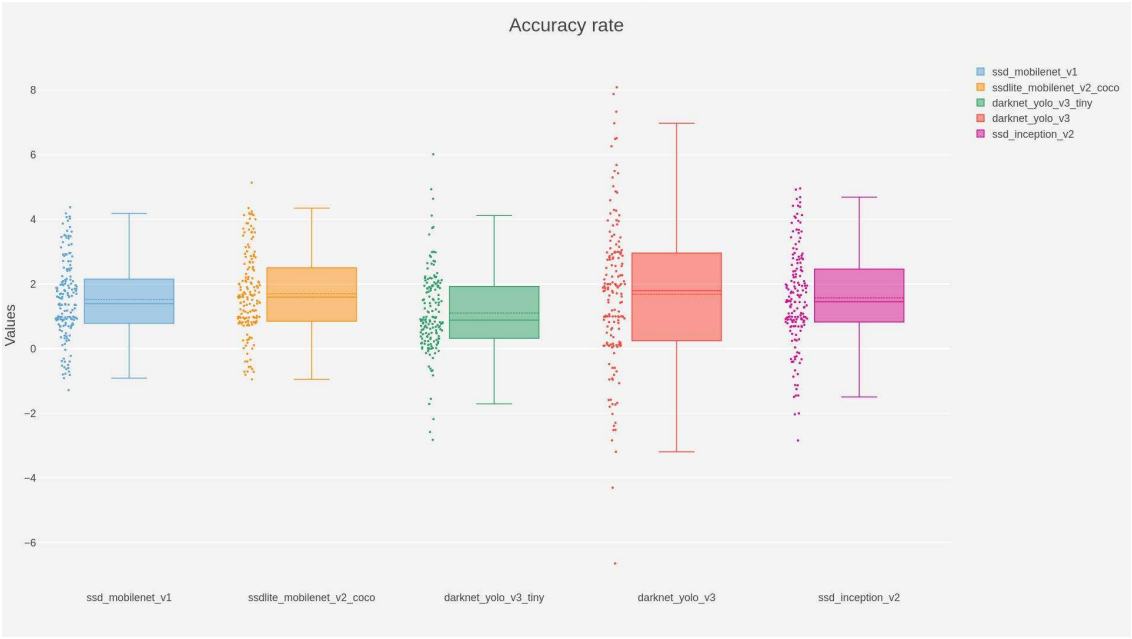
6. ábra. Detekció ideje másodpercben



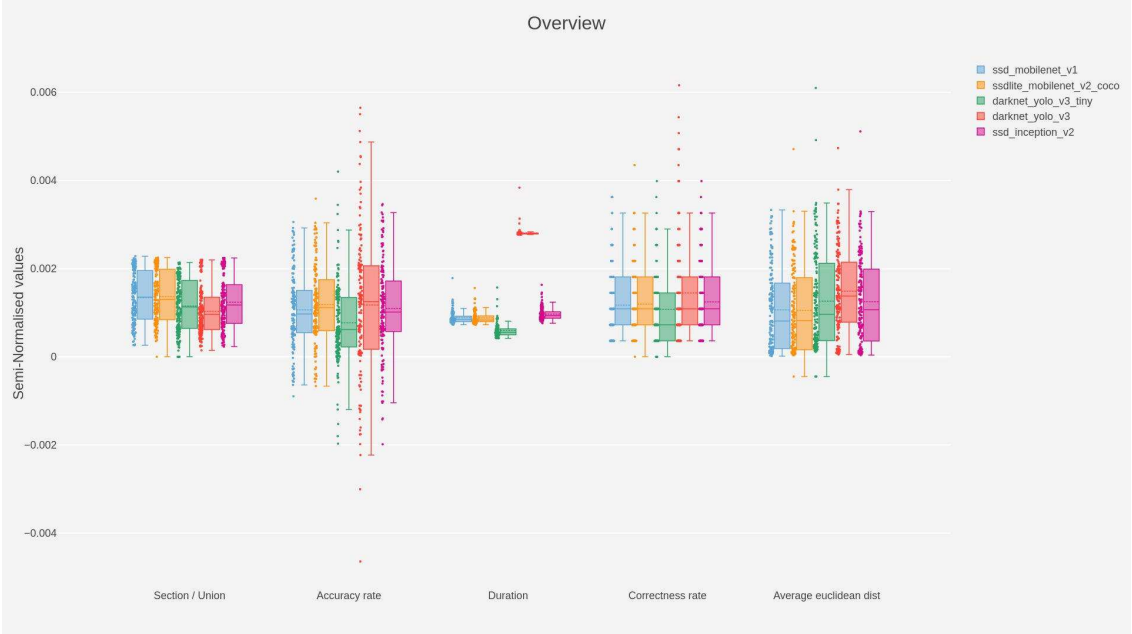
7. ábra. Helyességi ráta



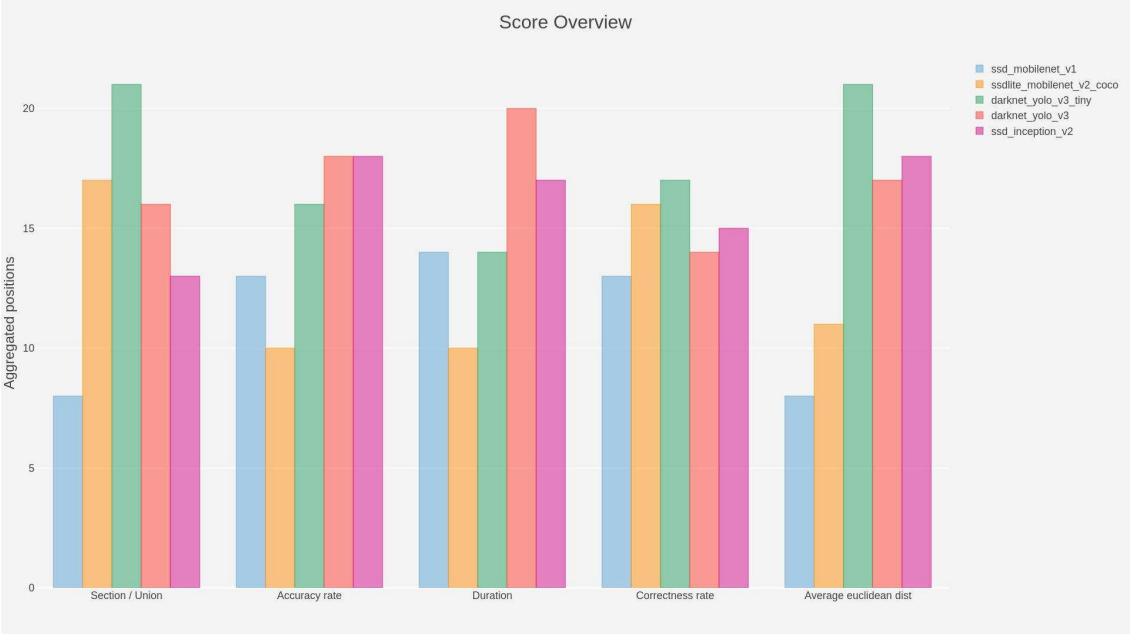
8. ábra. Pontossági ráta



9. ábra. A metrikák normalizált koordináta rendszerben



10. ábra. Súlyozott összesített helyezések metrikánként



3. Fejlesztői dokumentáció

3.1. Specifikáció

3.1.1. A probléma

Ahány cég, annyi különböző célkitűzés. Ez az olyan cégeknél sincs másképp amelyek objektumfelismeréssel foglalkoznak. Ahhoz, hogy bármely ilyen cég számára hasznossá váljon a keretrendszer, amelyet készítettem kellőképp adaptívnak kell lennie az adott feladathoz. Ennek köszönhetően tetszőleges metrikákkal, statisztikákkal és nyílt forráskódú objektumfelismerőkkel kell hogy tudjon operálni. Ezért a fejlesztés során OOP és funkcionális paradigmákat kellett követnem.

Az ilyen mérésekhez átlalában elég nehéz értékes annotációkat gyűjteni melynek két fő oka van. Az egyik lehetséges ok, hogy a cégek saját maguk és nagy fáradtságok árán jutottak hozzá (pl. felvettek több alkalmazotatt hogy kézzel végezzék el pixelenként), melynek köszönhetően nem áll szándékukban megosztani azt a publikummal. A másik lehetséges indok lehet az adott annotáció minősége. Gyakran csak adott specifikus szoftverrel lehet őket használni a kiterjesztéstük miatt, vagy pedig nagyon rosszul struktúráltak maguk a fájlok, és nehéz kinyerni belőlük a fontos paramétereket. Ezért körültekintőnek és kitartónak kell lenni a keresésükkor. Végülis találtam egy jó adatbázist, melyben 164 kép található.

A felhasználói dokumentációban említettem, hogy a detektorok közti sorrend kiszámításnál súlyozást használok. Habár ezek a súlyok a felhasználói oldalról nem állíthatóak a fejlesztői oldalról, a kódon belül igen. A jelenlegi súlyok kiválasztásának oka pusztán a szubjektív megítélésnek köszönhető, én úgy éreztem egymáshoz képest, a metrikák ennyire fajsúlyosak egymáshoz képest.

3.1.2. Megközelítés

Az imént említett generikus probléma megoldása során a korábban említett OOP programozás eszközeihez nyúltam. Az osztályok nagy részét örököltetéssel általánosítattam, leimplementálva a közös "core" funkciókat az ősosztályban. Így ha esetleg más szeretne készíteni egy új annotációs kezelő osztályt, akkor csak a specifikus részeket kell megírnia hozzá.

Másrészről funkcionális elemek is vannak benne, hiszen a program több része során tetszőleges függvényeket – metrikákat és statisztikákat - használok részszámolások során. Az ilyen helyzetekben a megvalósítás semmilyen szempontból nem használja ki az adott függvény belső tulajdonságait, paramétereit. Ennek köszönhetően bármely fejlesztőnek elég csak a specifikus részeket átírni, hozzáadni.

3.2. A felhasznált módszerek, fogalmak

3.2.1. A képek feldolgozása

A mérés során mért képeket egy publikus adatbázisból gyűjtöttem össze, melyeknek mérete 256x256. A feldolgozásukat a vizuális tartalmakon végzett hatékony műveletekre optimalizált OpenCV könyvtárral végeztem.

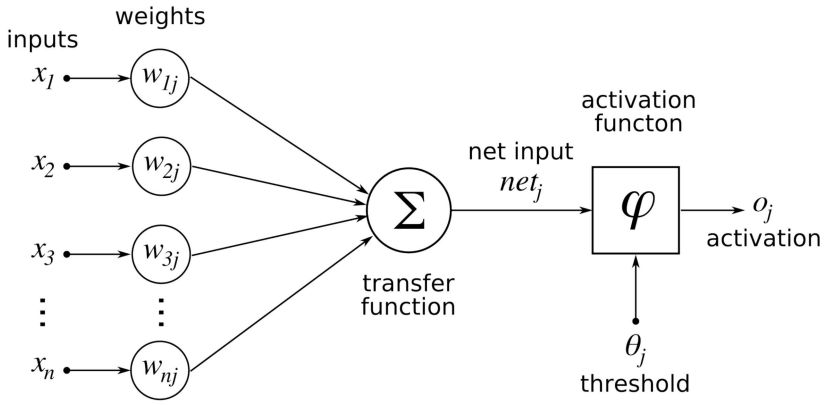
3.2.2. Az annotációk feldolgozása

Az annotációk egy-egy szöveges (txt) fájlban voltak megadva. Mivel az adatbázis relatíve régi, ezért volt pár felesleges megjegyzés is a fájlokban, mint például az elkészítésének ideje. Nekem az annotációkból csak az objektumok 2D-s elhelyezkedésére és típusára volt szükségem. Szerencsére ezek az információk mindig ugyanazokban a maradékosztály sorokban voltak. A szöveges sorokból regexp segítségével nyertem ki a fontos paramétereket.

3.2.3. A detektorok

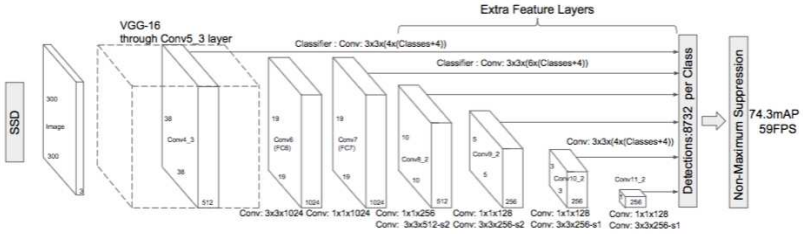
A kiértékelés során felhasznált detektorok mind neurális hálóknak egy speciális típusa. A neurális háló egy olyan függvény, mely sok nem-lineáris függvényből mint rétegből és együtthatóból mint súlyokból áll. A hálóknak az együtthatóit tanítás során szokták optimalizálni, úgy hogy a lehető legpontosabb predikciót adja a tanító adathalmaz során elsajátított információk segítségével. Az optimalizáláskor úgynevezett gradiens módszert használták.

11. ábra. Neurális háló

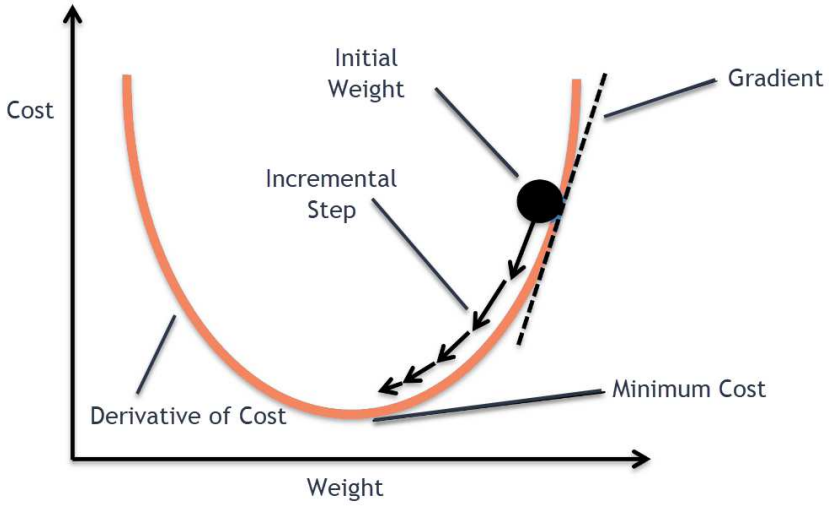


Mivel az eszköz amelyre fejlesztettem egy GPU, ezért a betanult háló súlyai tovább lettek optimalizálva a TensorRT könyvtár segítségével. A folyamat során a háló úgymond lefagyasztásra került, és a függvénykompozíciós reprezentáció áttért gráfra. A gráf éleinek módosításával és a súlyok mantisszájának csökkenésével a pontosság kis mértékű rovására jelentős gyorsaságbeli performancia növekedés érhető el.

12. ábra. Objektum felismerő - SSD



13. ábra. Gradiens módszer



3.2.4. Az IOU kiszámítása

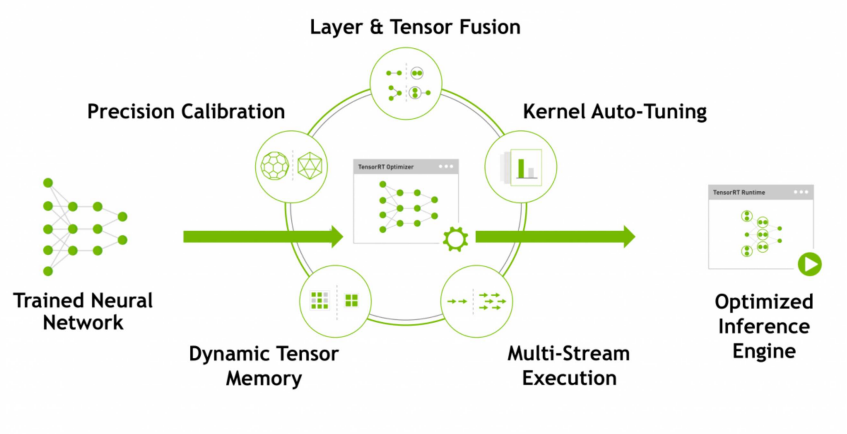
Az IOU-t (más néven Jaccard index) a 3.1-es algoritmussal lehet kiszámolni, feltéve ha a téglalapok oldalai merőlegesek az x,y tengelyekre.

Algorithm 3.1 IOU

Require: axMin, ayMin, axMax, ayMax;
Require: dxMin, dyMin, dxMax, dyMax;
Ensure: Jaccard Index

1: $xMin := max(axMin, dxMin)$
2: $yMin := max(ayMin, dyMin)$
3: $xMax := min(axMax, dxMax)$
4: $yMax := min(ayMax, dyMax)$
5: $aArea := (axMax - axMin) * (ayMax - ayMin)$
6: $dArea = (dxMax - dxMin) * (dyMax - dyMin)$
7: $intersection = max(0, xMax - xMin) * max(0, yMax - yMin)$
8: **return** $intersection/(aArea + dArea - intersection)$

14. ábra. TensorRT



3.2.5. Vizualizáció

A vizualizációhoz a Plotly nevezetű könyvtárat használtam, mely pythonon kívül számos egyéb programozási nyelvhez is elérhető. Segítségével a diagramokat html formátumban lehet megtekinteni interaktív módon.

3.3. A program logikai és fizikai szerkezete

3.3.1. A program fejlesztői dokumentációja

ResourceHandler
Absztrakt erőforráskezelő osztály

- ___init___
  Beállítja az erőforráshoz tartozó könvytárat és inicializálja a következő erőforrás indexét
  - Kulcsszavas argumentumok
    - directory {string}: Az erőforrás által használt fájlok helye
- _getNextResourceName
  Visszaadja a következő erőforrás nevét, ha van ilyen.
  - Visszatérési érték: {string|None} - Az erőforrás neve
- getResource
  Visszaadja a következő erőforrást, ha van ilyen.
  - Kulcsszavas argumentumok
    - resName {string|None}: Az erőforrás neve
  - Visszatérési érték: {file} - Maga az erőforrás

- reset
  Visszaállítja az erőforrás indexet 0-ra.

ImageHandler
Képkezelő osztály, a ResourceHandler leszármazottja. Elvégzi a képek betöltését adott könyvtárból.

- _handleResource
  Visszaadja a következő képet
  - Argumentumok
    - imageName {string}: A kép neve
  - Visszatérési érték: {array} - Maga a kép
- getFile
  Visszaadja adott kép nevéhez a teljes elérési utat.
  - Argumentumok
    - imageName {string|None}: A kép neve
  - Visszatérési érték: {string} - Az útvonal
- reset
  Visszaállítja az erőforrás indexet 0-ra.

AnnotationHandler
Absztrakt annotációkezelő osztály, a ResourceHandler leszármazottja.

- _handleResource
  Megnyitja az annotációs fájlt, majd kinyeri a pontos annotációs információt
  - Argumentumok
    - annotation {string}: Az annotációs fájl neve
  - Visszatérési érték: {dict} - Az annotációk

Pfp
PenFudanPed annotációs adatbázis kezelő osztály, az AnnotationHandler leszármazottja. Elvégzi a PenFudanPed típusú annotációs fájlok betöltését, és parseolását regexp. segítségével adott könyvtárból.

- _extractInfo
  Kinyeri az annotációs adatokat egy fájlból

  - Argumentumok
    * data {array}: Az annotációs fájl tartalma sorokra tördelve
  - Visszatérési érték: {dict} - Az annotációk fileName, dimensions, object annotations-re tördelve

Detector
Absztrakt objektumfelismerő osztály

- ___init___
  Konstruktor
  - Kulcsszavas argumentumok
    * model {string}: Az objektumfelismeréshez használt model neve
    * confTh {float}: A konfidencia küszöb annak eldöntésére elvetjük-e a detektált objektumot

DarknetDetector
Darknet objektumfelismerő osztály, a Detector leszármazottja. Betölti a modellt memóriába, illetve elvégzi az objektumok felismerését adott képen.

- ___init___
  Konstruktor
  - Kulcsszavas argumentumok
    * model {string}: Az objektumfelismeréshez használt model neve
    * confTh {float}: A konfidencia küszöb annak eldöntésére elvetjük-e a detektált objektumot
    * metaPath {string}: A metadatahoz tartozó elérési útvonal
- loadModel
  Betölti a szükséges C függvényeket és adatokat
- process
  Elvégzi a detekciót egy adott képen
  - Argumentumok
    * image {array}: Az adott kép
  - Visszatérési érték: {tuple} - A detekció mért ideje és a felismert objektumok

- _getBoxPoints
  Kiszámítja a felismert objektum közepéből és a méretéből a bal felső és jobb alsó x,y koordinátákat.
  - Argumentumok
    - box {array}: Az objektum közepe és mérete lebegőpontos számokkal (x,y,w,h)
  - Visszatérési érték: {array} - (xmin,ymin,xMax,yMax)

TfDetector
Tensorflow objektumfelismerő osztály, a Detector leszármazottja. Letölti az adott modellhez szükséges fájlokat internetről, elkészíti hozzá a hálót, betölti a modellt memóriába, illetve elvégzi az objektumok felismerését adott képen.

- ___init___
  Konstruktor
  - Kulcsszavas argumentumok
    - model {string}: Az objektumfelismeréshez használt model neve
    - doBuild {bool}: Beállítja kell-e buildelni a modelt protobuf fájlból
    - confTh {float}: A konfidencia küszöb annak eldöntésére elvetjük-e a detektált objektumot
    - detectorType {string}: A tensorflowos objektumfelismerő típusa
    - labelMap {string}: Az objektumok neveit tartalmazó fájl elérési útja
- _buildModel
  Létrehozza a neurális hálót, miután letöltötte a súlyokat és a rétegeket, majd kimenti egy protobuf fájlba
- _buildGraph
  Létrehozza a neurális hálót
  - Argumentumok
    - confPath {string}: A háló rétegeit és súlyait tartalmazó fájl elérési útja
    - checkPoint {string}: Az utolsó buildelt verzió elérési útja
- _getModelUrl
  Visszaadja a model nevének megfelelő config és checkpoint url-eket
  - Visszatérési érték: {dict} - (config url, checkpoint url)

- _downloadModel
  Letölti a config és checkpoint fájlokat
  - Kulcsszavas argumentumok
    - downloadDir {string}: A model letöltésének helye
  - Visszatérési érték: {tuple} - (config, checkpoint)
- _readLabelMap
  Betölti az objektumcímkéket a egy osztályváltozóba a labelMap-ból
  - Visszatérési érték: {dict} - A címkék
- loadModel
  Betölti a háló súlyait, rétegeit a RAM-ba
- _loadGraph
  Betölti a hálót a protobuf fájlból
  - Visszatérési érték: {graph} - A tensorflow gráf(háló) objektum
- process
  Elvégzi a detekciót egy adott képen
  - Argumentumok
    - originalImg {array}: Maga az eredeti kép
  - Visszatérési érték: {tuple} - (időtartam, a felismert objektumok)
- _postprocess
  Felskálázza a kinyert pozíciókat a kép méretével, elveti azokat az objektumokat amelyek konfidenciaértéke nem üti meg küszöbértéket
  - Argumentumok
    - img {array}: A kép
    - boxes {array}: A felismert objektumokhoz tartozó téglalapok
    - scores {array}: A konfidenciaértékek
    - classes {array}: Az objektumok nevei
  - Visszatérési érték: {tuple} - A elfogadott objektumok adatai, (téglalapok, konfidenciaértékek, osztályok)
- clean
  Jelzi a garbage collector számára, hogy a feleslegessé vált osztályváltozók törölhetők a memóriából.

Evaluator
Kiértékeli az adott model eredményességét egy adott képen a kapott metrikák és annotációk segítségével.

- eval
  Kiértékeli a metrikákat a felismert objektumokon és az annotációkon
  - Argumentumok
    - detector {Detector}: Az objektumfelismerő
    - img {array}: Maga a kép
    - annotations {array}: Az annotációk
    - metrics {dict}: A metrikákat tartalmazó szótár
  - Visszatérési érték: {dict} - A metrikák eredményei

BaseMetrics
Definiálja a korábban ismertetett metrikákat.

- getAll
  Visszaadja az összes definiált metrikát
  - Visszatérési érték: {dict} - A metrikák és paramétereik (súly, maximalizálni kell-e, a függvény)
- minDist
  Visszaad egy olyan függvényt, mely kiszámolja az adott téglalap és a paraméterként várt annotáció távolságát egyes norma szerint
  - Argumentumok
    - detection {dict}: A felismert objektum
  - Visszatérési érték: {function} - Az egyes norma függvény
- getClosest
  Meghatározza adott felismert objektum téglalapjához a hozzá legközelebbi annotációt egyes norma szerint
  - Argumentumok
    - detection {dict}: A felismert objektum
    - annotations {array}: Az annotációkat tartalmazó tömb
  - Kulcsszavas argumentumok
    - param {string|None}: Meghatározza, hogy az egész annotációt vagy annak csak az adott kulcsához tartozó értéket kell-e visszadni.

  - Visszatérési érték: {dict} - A legközelebbi annotáció adata

- euclideanDist
  Kiszámolja az átlagos euklideszi távolságot az egyes detekciók és a hozzájuk legközelebbi annotációk közt
  - Argumentumok
    * detections {array}: A detekciók
    * annotations {array}: Az annotációk
  - Visszatérési érték: {float} - Az átlagos távolság

- iou
  Kiszámolja a Jaccard indexet minden egyes detektorhoz
  - Argumentumok
    * detections {array}: A detekciók
    * annotations {array}: Az annotációk
  - Visszatérési érték: {float} - Jaccard Index

- getConfidentalAccuracy
  Kiszámolja a pontossági rátát
  - Argumentumok
    * detections {array}: A detekciók
    * annotations {array}: Az annotációk
  - Visszatérési érték: {float} - A ráta

- filterDetections
  Kiszűri a detekciók konfidenciaértékeit az alapján a hozzájuk legközelebbi annotáció objektumtípusával megegyeztek-e
  - Argumentumok
    * detections {array}: A detekciók
    * annotations {array}: Az annotációk
  - Visszatérési érték: {tuple} - (Jó konfidenciaértékek, Rossz konfidenciaértékek)

- getCountAccuracy
  Kiszámolja a helyességi rátát
  - Argumentumok

        * detections {array}: A detekciók
        * annotations {array}: Az annotációk
    - Visszatérési érték: {float} - A ráta

Main
A kiértékelés főosztálya, a program belépési pontja.

- ___init___
  Inicializálja az egyes osztályokat, mint privát tagokat.

- clean
  Jelzi a garbage collector számára, hogy a feleslegessé vált osztályváltozók törölhetők a memóriából.

- _parseArgs
  Beolvassa a parancssori argumentumokat

    - Visszatérési érték: {dict} - Az argumentumok beparsolva.

- _run
  Elvégzi a metrikák kiértékelését, majd elemzi az eredményeket

- _initEachRun
  Inicializája az eredményt szótárt az új detektor miatt

    - Argumentumok
        * results {dict}: Az eddigi eredmények
        * detName {string}: A detektor neve
    - Visszatérési érték: {dict} - Az új eredmények

- _appendResult
  Inicializája az eredményt szótárt az új detektor miatt

    - Argumentumok
        * results {dict}: Az eddigi eredmények
        * detName {string}: A detektor neve
        * res {dict}: Az utolsó metrikák általi kiértékelés eredménye
    - Visszatérési érték: {dict} - Az új eredmények

Aggregator
Összegzi az eredményeket különböző statisztikák szerint, majd meghatározza a detektorok közti sorrendet illetve vizuálisan is megjeleníti az eredményeket.

- ___init___
  Beállítja a statisztikákat, a statisztikák prioritását, a színeket a megjelenítéshez illetve a metrikák teljes nevét.

- _initAggr
  Inicializálja az összegzéshez használt struktúrát

  - Argumentumok

    * data {dict}: A korábbi kiértékelés eredménye

  - Visszatérési érték: {dict} - Az új reprezentáció

- aggregate
  Összegez, kiértékeli a statisztikákat, kiszámítja a sorrendet majd megjelenít

  - Argumentumok

    * results {dict}: A korábbi kiértékelés eredménye

- _simpleBoxPlot
  Rajzol egy-egy boxplotot a detektorok eredményeivel egyesével metrikánként

  - Argumentumok

    * results {dict}: A korábbi kiértékelés eredménye

- _groupedBoxPlot
  Rajzol egy összesített boxplotot a metrikákat csoportosítva

  - Argumentumok

    * results {dict}: A korábbi kiértékelés eredménye

- _rankingPlot
  Kirajzolja a detektorok súlyozott helyezéseit metrikánként

  - Argumentumok

    * results {dict}: A korábbi kiértékelés eredménye

- _display
  Kirajzolja az összes diagrammot

  - Argumentumok

        * results {dict}: A korábbi kiértékelés eredménye

- _calculateScores
  Kiszámítja a detektorok a különböző statisztikákban elért helyezések összegét súlyozva metrikánként

  – Argumentumok

    * data {dict}: A korábbi kiértékelés eredménye
    * detectors {array}: A detektorok nevei

  – Visszatérési érték: {dict} - A helyezések súlyozott összege

- _orderByScores
  Kiszámítja a detektorok egy adott metrikában elért a különböző statisztikákban elért helyezések összegét

  – Argumentumok

    * results {dict}: A korábbi kiértékelés eredménye
    * metric {array}: A metrika neve

  – Visszatérési érték: {dict} - A helyezések összege

- _calculateOrder
  Kiszámítja a detektorok közti sorrendet, majd kimenti az eredményeket egy szöveges fájlba

  – Argumentumok

    * data {dict}: A korábbi kiértékelés eredménye
    * detectors {array}: A detektorok nevei

- _saveResults
  Kimenti a detektorok közti sorrendet szöveges fájlba

  – Argumentumok

    * scores {dict}: A detektorok összesített helyezése
    * order {array}: A detektorok, helyezés alapján növekvő sorrendben

Visualiser
Rajzolásért, az eredmények megjelenítéséért felelős osztály.

- ___init___
  Konstruktor

- addTrace
  Eltárolja a kirajzolandó elem adatait lokálisan
  - Argumentumok
    * x {array}: Az X értékek
    * y {array}: Az Y értékek
    * name {str}: A kategória/csoport neve
    * color {str}: Az elem színe
- _setBoxTrace
  A kapott elem adatai alapján előállít egy boxplot objektumot
  - Argumentumok
    * x {array}: Az X értékek
    * y {array}: Az Y értékek
    * name {str}: A kategória/csoport neve
    * color {str}: Az elem színe
  - Visszatérési érték: {Box} - Boxplot objektum
- _setBarTrace
  A kapott elem adatai alapján előállít egy oszlopdiagram objektumot
  - Argumentumok
    * x {array}: Az X értékek
    * y {array}: Az Y értékek
    * name {str}: A kategória/csoport neve
    * color {str}: Az elem színe
  - Visszatérési érték: {Box} - Oszlopdiagram objektum
- _getPlotParams
  Előállítja a rajzoláshoz szükséges adatokat a kapott paraméterek alapján
  - Argumentumok
    * mode {string|None}: csoportosan szeretnénk-e rajzolni
    * chartType {str}: Oszlop vagy box
  - Visszatérési érték: {tuple} - (box-e, oszlop-e, diagram előállító függvény)
- plotChart
  Kirajzol egy diagramot a kapott paraméterek alapján

- Argumentumok
  - fileName {string}: A
  - title {string}: A diagram címe
  - yaxis {string}: Az y tengely címe
- Kulcsszavas argumentumok
  - xaxis {string|None}: Az x tengely címe
  - mode {string|None}: Sima, vagy csoportos
  - chartType {string}: Box vagy oszlop
  - resolution {tuple}: A tárolandó kép felbontása

3.3.2. A program szerkezete

A teljes programot végülis egy fájlban hagytam, a *main.py*-ban, hiszen megítélésem szerint nem volt olyan nagy terjedelmű a kód, amely a program modularizálást feltétlen szükségessé tette volna.

A program logikai szerkezetének leírásához UML diagrammokat készítettem.

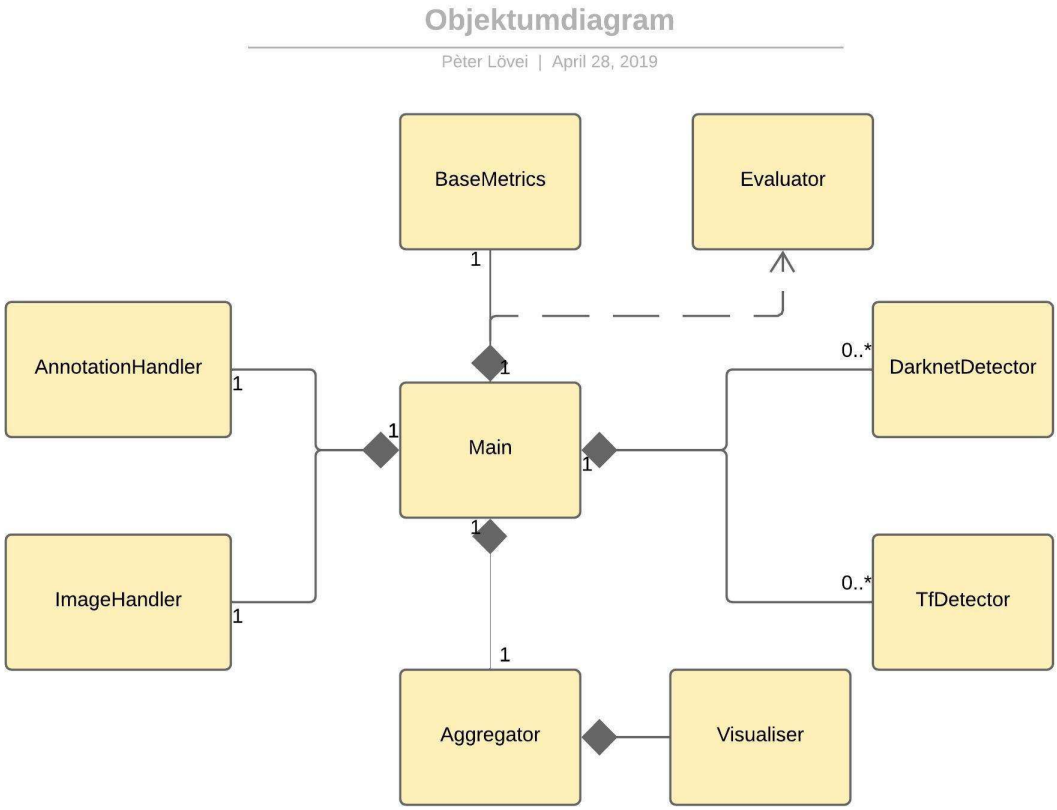
A 15. ábrán azt láthatjuk, hogy az egyes objektumok hogyan tartalmazzák egymást. Mint mefigyelhetjük, a Main a fő objektum, ő rendelkezik számos más objektummal. A kapcsolat sötét rombusz oldali vége jelzi, hogy az adott osztályban található, a számmal elátott vége pedig, hogy milyen objektumból és mekkora számossággal. A 0..* jelzi, hogy ott tetszőleges számú tartalmazásról van szó. Szagatott vonallal jelöltem egyfajta dependenciát, mely a tartalmazás helyett, az egyik objektum függését jelzi egy másik objektumtól.

17., 16., 18. ábrákon a program struktúráját láthatjuk osztálydiagrammokon keresztül. A "c" jelöli az osztály nevét, "m" a metódusok nevét, "f" pedig az adattagokét. A kék nyilak jelzik az öröklődés irányát, tehát ahova mutatnak az az ősosztály, míg ahonnan mutatnak az a leszármazott. A metódusok nevei mellett zárójelben az elvárt paraméterek (és azok esetleges alap értékük) találhatók. A _-al kezdődő metódusok és adattagok jelölik, hogy azok az adott osztály privát tagjai. (Mivel pythonban nincs ilyen jellegű enkapszuláció, ezért ez csak jelképes elfedés.)

A 19. ábrán a program szekvenciadiagrammja látható. A függőleges vonalak jelzik az egyes résztvevők élettartamát. A bal oldali az aktor, aki végzi az interakciót, a többi pedig a program egyes komponensei, objektumai. A felhasználónak nem sok interakciója van azon felül, hogy futtatja a programot, hiszen ezután csak hosszas számolás után a végeredményt kapja vissza. A programban két egymásba ágyazott ciklus található, melyek az egyes detektorokra és kiértékelésre vonatkoznak. Míg teli nyíl jezli az "üzenet"[5] küldését, addig a szagatott a korábbi üzenetre való választ.

[5]Azaz egy methódushívást.

15. ábra. Objektumok



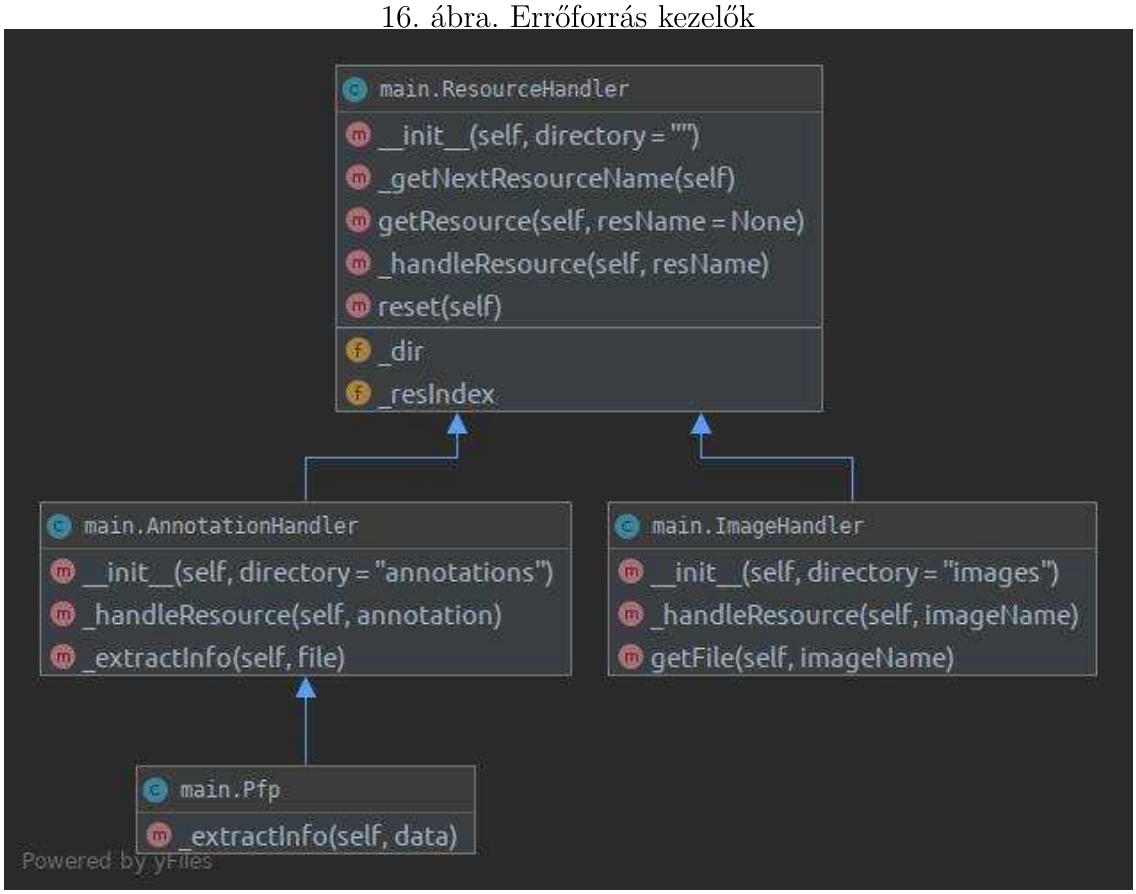
3.3.3. Adatstruktúrák

A detektorok kiértékeléséhez használt struktúra

A detektorok kiértékeléséhez egy egymásba ágyazott dictionarykből álló struktúrát használtam, amelyet egy json struktúrával prezentálok a szakdolgozatomban a könnyebb áttekinthetőség érdekében. Az _n, _m jelölik, hogy az adott kulcs fajtából több is szerepel az adott szinten, nevükben különbözve. A struktúra így fest:

```
{
  "detektor_n" (pl. ssd-mobilenet-v1):  {
    "metrika_m" (pl. iou): {
      "suly": lebegopontos szam,
      "maximalizalando-e": igen/nem,
      "meresi eredmenyek": egy lista
    }
  }
}
```

16. ábra. Errőforrás kezelők



A helyezések kiértékeléséhez használt struktúra

```
{
  "metrika_m":  {
    "suly": lebegopontos szam,
    "maximalizalando-e": igen/nem,
    "statisztikak": {
      "stat_k" (pl. atlag): {
        "detektor_n": lebegopontos szam
      }
    },
    "meresi eredmenyek": {
      "detektor_n": lista
    }
  }
}
```

3.4. Hibakezelés

A programomhoz készítettem hibakezelést is, mely a felhasználói iterakció során előforduló esetleges hibák előfordulásait hivatott ellenőrizni potenciálisan figyelmeztetve erre a felhasználót. Habár a programomnak csak egy publikus belépési pontja van a felhasználó szemszögéből - nevezetesen a terminál - úgy döntöttem, hogy minden olyan osztályom publikus függvényeinél kezelem a hibát, ahol azt a paramétereket szükségessé tehetik.

A hiba elkapásához a python beépített *if* és *try - except* konstrukcióit használtam, a hiba jelzésére pedig az ugyancsak beépített *raise ValueError, raise TypeError* függvényeit használtam.

3.5. Tesztelés

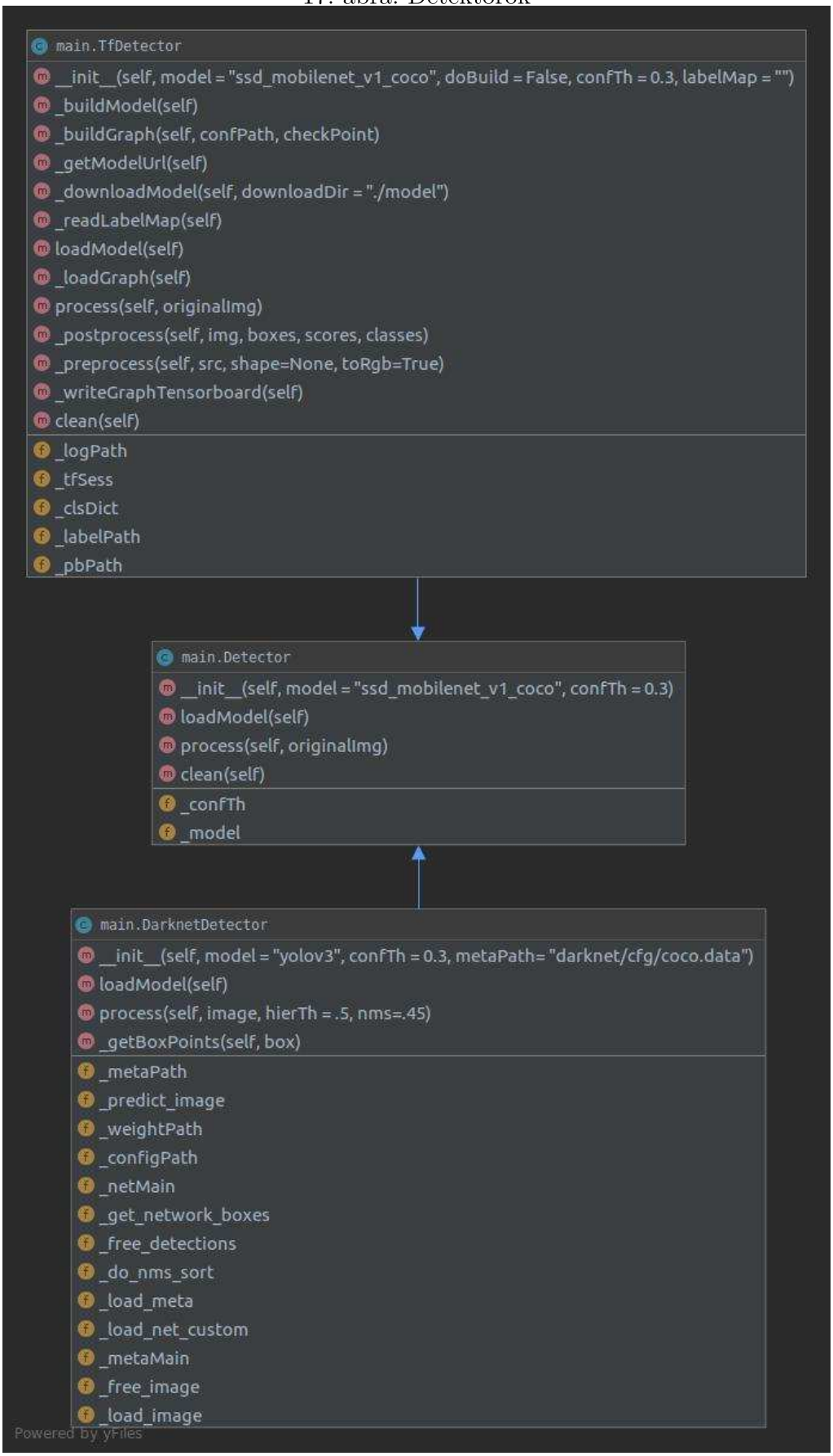
3.5.1. A tesztelési terv

A teszteléshez készítettem egy másik python fájlt, melynek a neve *test.py*. Ebben a fájlban a különböző osztályok metódusainak teszteléséhez unittesteket készítettem. A legtöbb publikus függvényt teszteltem, illetve azokat a privát metódusokat, melyeket úgy véltem jelentősen átalakítják a program futása során használt adatok struktúráját. A tesztelés során különös hangsúlyt fektettem azokra a metódusokra, melyek függvényként, azaz adott bemenethez adott kimenetet rendelnek. Ezen felül külön vizsgáltam feketedoboz- és fehérdoboz tesztestes

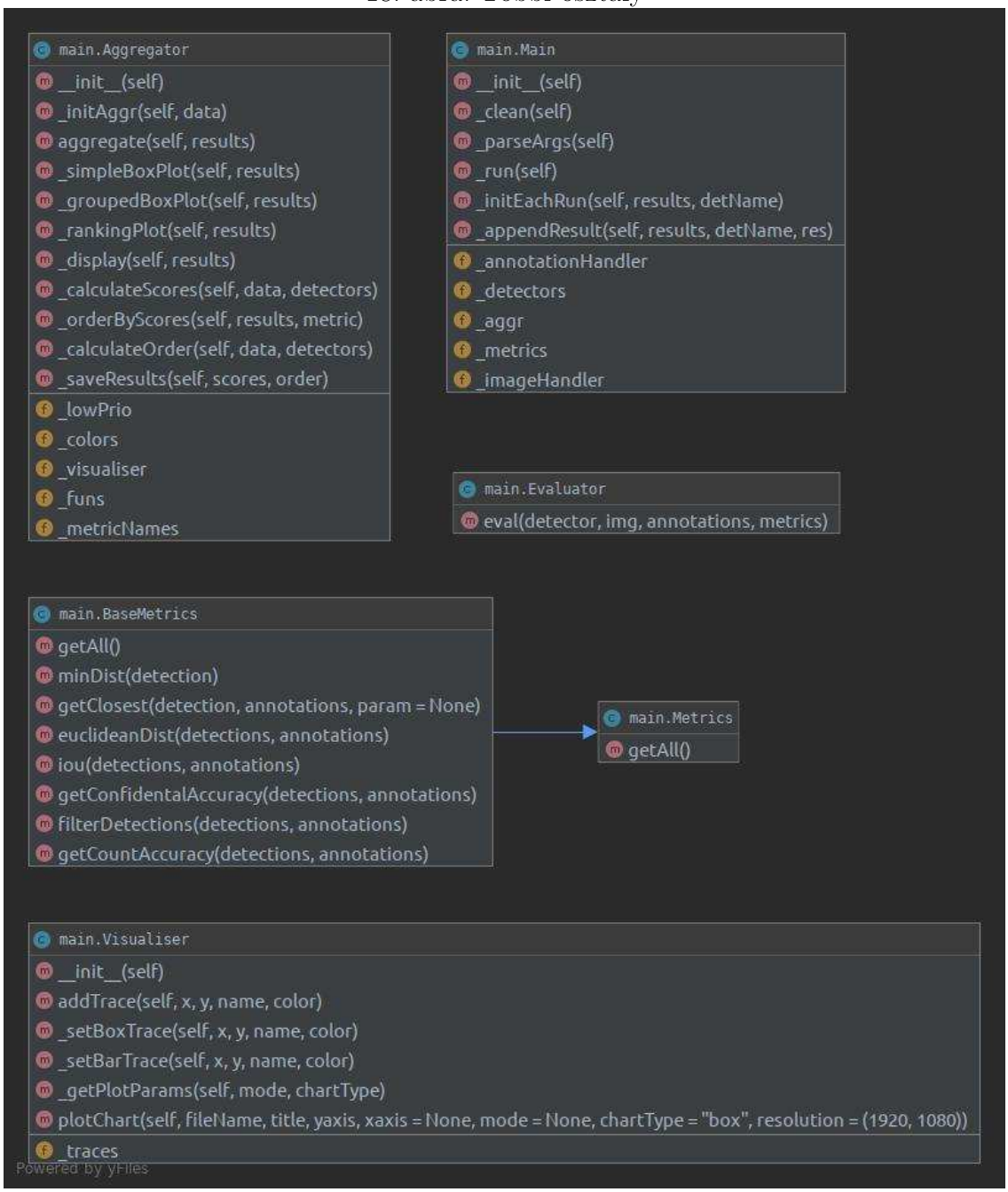
3.5.2. A tesztelés eredménye

A tesztelés során összesen 28 unittesztet írtam, melyek mind sikerrel zárultak. Erről a *test.py* lefuttatása után meg is győződhetünk.

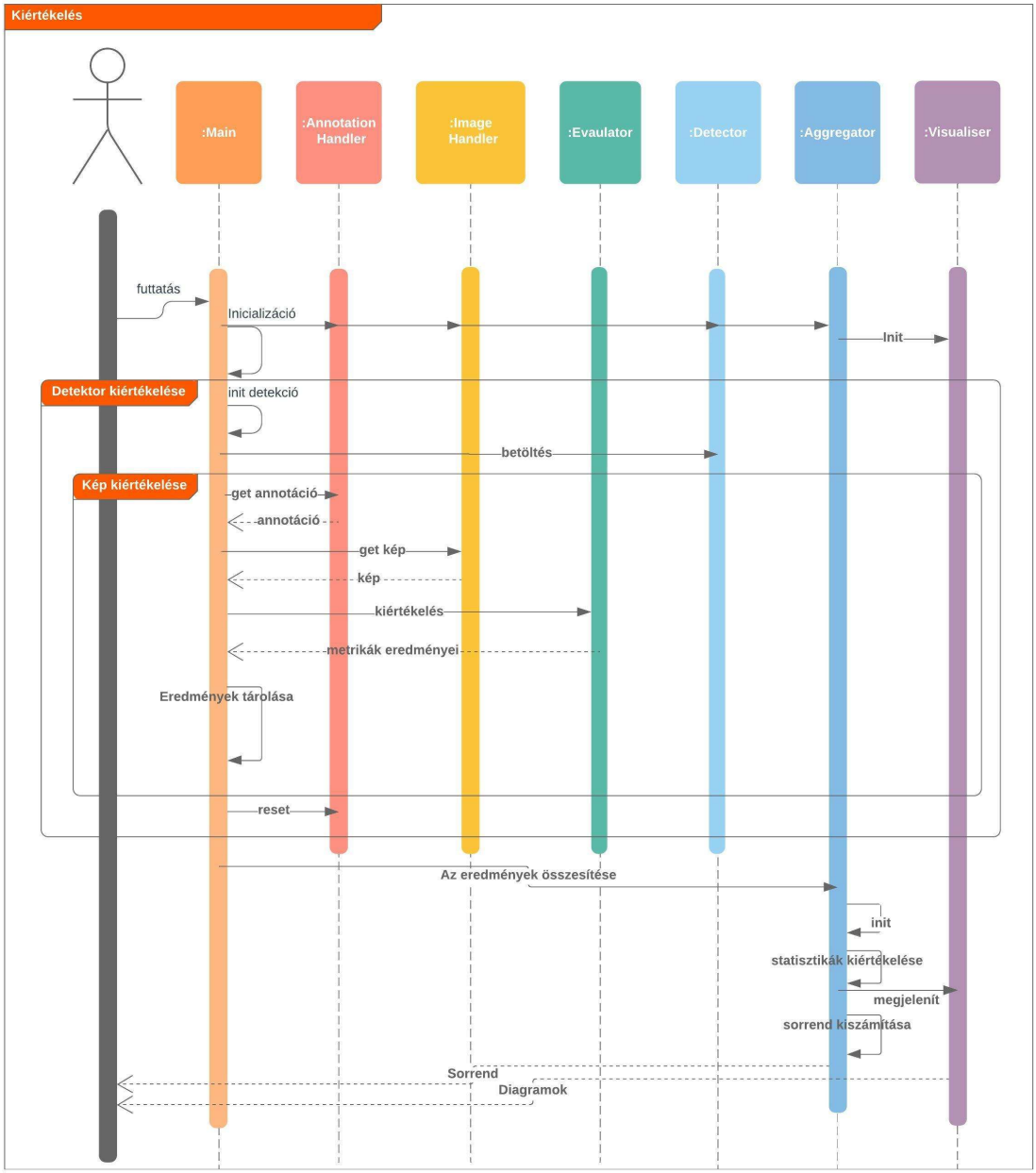
17. ábra. Detektorok



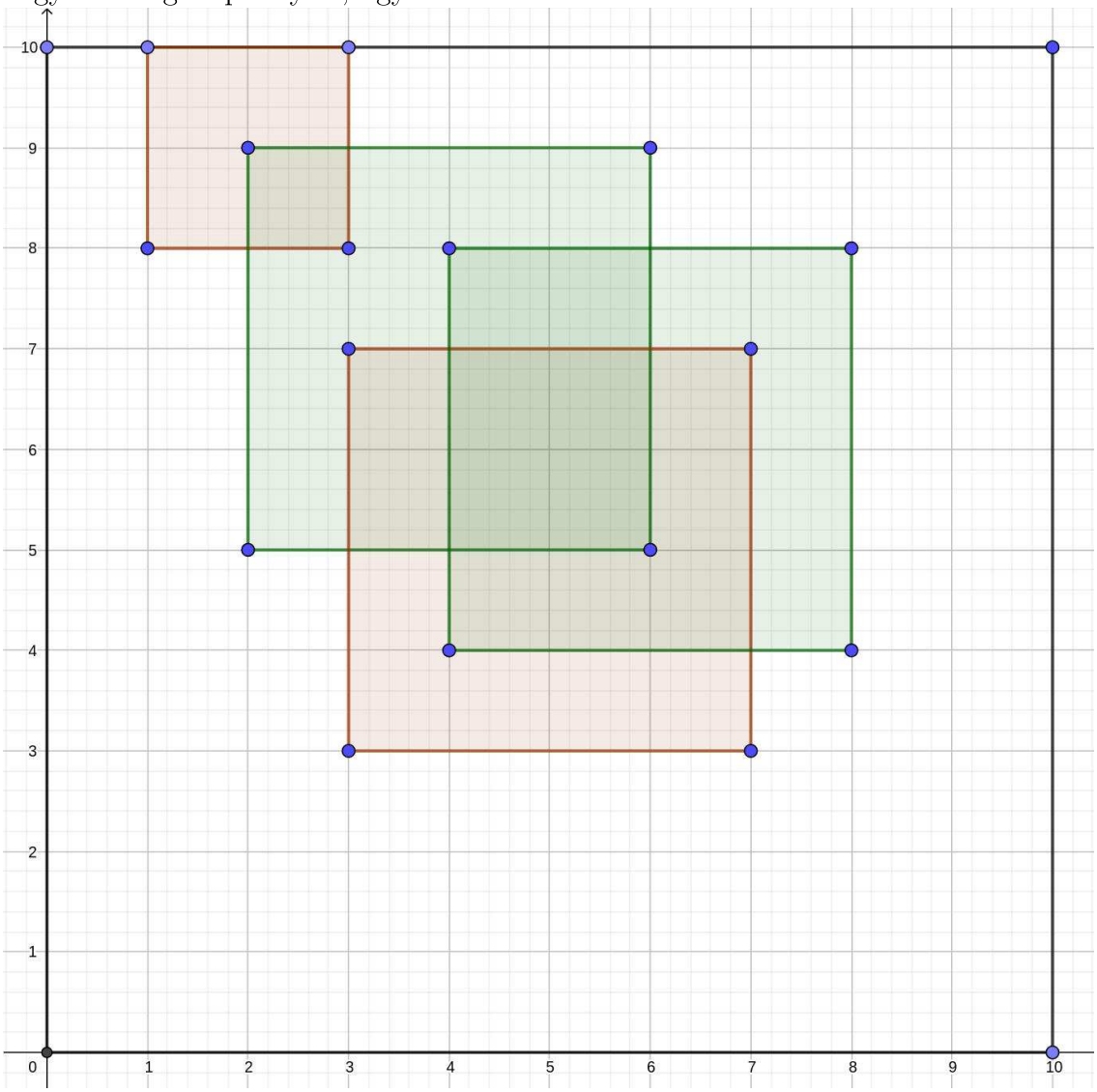
18. ábra. Többi osztály



19. ábra. A program szekvenciája



20. ábra. A metrikák teszteléséhez használt téglalapok(négyzetek) elhelyezkedése szemléletesen. A (0,0) - (10,10) pontok által kifeszített síkidom mind egy detekció, mind egy annotációhoz tartozik. Ezen felül a zöld színű négyzetek jelölik a detekciókat, a narancssárgák pedig az annotációkat. Az egyszerűség kedvéért használtam négyzetet téglalap helyett, ugyanis ezt nincs hatással a metrikák működésére.



21. ábra. A tesztelés eredménye

```
....
----------------------------------------------------------------------
Ran 28 tests in 162.699s

OK
```

4. Irodalomjegyzék

[1] Robert J. Wang, Xiang Li és Charles X. Ling: A Real-Time Object Detection System on Mobile Devices, Advances in Neural Information Processing Systems, 31, 2018 [10].

[2] Móri F. Tamás, Szeidl László és Zempléni András: Matematikai statisztika példatár, ELTE Eötvös, 1997 [228], ISBN-978-1-4503-4444-9.

[3] Joseph Redmon: Darknet: Open Source Neural Networks in C, (2013–2016), `http://pjreddie.com/darknet/` [2019. május 13..]

[4] Kanokwan Rungsuptaweekoon, Vasaka Visoottiviseth és Ryousei Takano: Evaluating the power efficiency of deep learning inference on embedded GPU systems, 2nd International Conference on Information Technology, 17505433, 2017 [1-5].